`std::vector<T>`

- Динамический массив элементов типа `T`
- Обращение к элементу за `O(1)`
- Добавление в конец за `O(1)`
- Удаление последнего элемента за `O(1)`
- Добавление в начало, середину за `O(n)`
- Удаление из начала, середины за `O(n)`
- Поиск элемента за `O(n)`
- `#include <vector>`

std::vector<T>

```cpp
std::vector<int> numbers;
numbers.push_back(1); // numbers: [1]
numbers.push_back(3); // numbers: [1, 3]

numbers[0] = 4;  // numbers: [4, 3]

std::cout << numbers.size() << std::endl; // 2, за O(1)
std::cout << numbers.back() << std::endl; // 4, numbers[numbers.size() - 1]
std::cout << numbers.front() << std::endl; // 4, numbers[0]

numbers.pop_back();

std::cout << numbers.size() << std::endl; // 1

std::cout << numbers.empty() << std::endl; // 0

std::cout << std::boolalpha << numbers.empty() << std::endl; // false

numbers.pop_back();

std::cout << numbers.empty() << std::endl; // 1

std::cout << std::boolalpha << numbers.empty() << std::endl; // true
```

Итерация по вектору

```
for (const auto x : std::vector<int>{1, 2, 3, 4}) {
    std::cout << x << std::endl;
}
```

`std::string`

- Разбирали
- Можно сказать, что это `std::vector<char>` с дополнительными методами, специфичными для строки
- `#include <string>`

std::pair<T1, T2>

```
namespace std {

template<F, S>
struct pair {
    F first;
    S second;
};

} // namespace std
```

Tuple

- Кортеж

```
std::tuple<int, int> foo_tuple()
{
    return {1, -1};  // Error until N4387
    return std::tuple<int, int>{1, -1}; // Always works
    return std::make_tuple(1, -1); // Always works
}

int main() {
    const auto f = foo_tuple();
    std::cout << std::get<1>(f) << std::endl; // -1
}
```

`std::unordered_map<K, T>`

- Отображение из объектов типа `K` в объекты типа `T`
- Хеш-таблица
- Поиск по ключу, добавление, удаление за `O(1)`
- `#include <unordered_map>`

std::unordered_map<K, T>

```
std::unordered_map<std::string, int> map;
map["abc"] = 1;
map["c"] = 5;

std::cout << map["abc"] == 1 << std::endl; // 1
std::cout << map["c"] == 5 << std::endl; // 1

const std::string str("k");
map.emplace(str, 32);

if (map.contains(str)) {
    std::cout << str << ": " << map[str] << std::endl;
}

std::cout << map.size() << std::endl; // 3
map.erase(str);
std::cout << map.size() << std::endl; // 2
```

Итерация по `std::unordered_map<K, T>`

```
const auto map = std::unordered_map<int, int>{
    {1, 2},
    {3, 4}
};
for (const auto pair : std::unordered_map<int>{1, 2, 3, 4}) {
    std::cout << pair.first << ' ' << pair.second << std::endl;
}
```

Итерация по `std::unordered_map<K, T>`

```
const auto map = std::unordered_map<int, int>{
    {1, 2},
    {3, 4}
};
for (const auto [k, v] : std::unordered_map<int>{1, 2, 3, 4}) {
    std::cout << k << ' ' << v << std::endl;
}
```

Structured binding

```
int a[2] = {1, 2};

auto [x, y] = a;     // creates e[2], copies a into e,
                     // then x refers to e[0], y refers to e[1]
auto& [xr, yr] = a; // xr refers to a[0], yr refers to a[1]
```

Structured binding

```
float x{};
char  y{};
int   z{};

std::tuple<float, char, int> tpl(x, y, z);
const auto& [a, b, c] = tpl;
```

Structured binding

```
struct S
{
    mutable int x1;
    double y1;
};

S f() { return S{1, 2.3}; }
int main()
{
    const auto [x, y] = f();
    std::cout << x << ' ' << y << '\n';  // 1 2.3
    x = -2;   // OK
//  y = -2.;  // Error: y is const-qualified
    std::cout << x << ' ' << y << '\n';  // -2 2.3
}
```

`std::unordered_set<T>`

- Множество объектов типа `T`
- Поиск, добавление, удаление за `O(1)`

std::unordered_set<T>

```
std::unordered_set<int> mySet{2, 7, 1, 8, 2, 8};
print(mySet);

mySet.insert(5);
print(mySet);

if (auto iter = mySet.find(5); iter != mySet.end())
    mySet.erase(iter);
print(mySet);

mySet.erase(7);
print(mySet);
```

Итерация по `std::unordered_set<T>`

```
for (const auto x : std::unordered_set<int>{1, 2, 3, 4}) {
    std::cout << x << std::endl;
}
```

`std::map<K, V>, std::set<K>`

- красно-черное дерево (сбалансированное дерево)
- Поиск, добавление, удаление за `O(log(n))`
- Гарантия отсортированности объектов при итерации по данным контейнерам
- `.front()` самый маленький элемент в контейнере
- `.back()` самый большой элемент в контейнере